Lazy Mobile Intruders

Mödersheim, Sebastian Alexander; Nielson, Flemming; Nielson, Hanne Riis

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Lazy Mobile Intruders *

Sebastian Mödersheim, Flemming Nielson, and Hanne Riis Nielson

DTU Informatics
IMM-Tech-Report 2012-13

Abstract. We present a new technique for analyzing platforms that execute potentially malicious code, such as web-browsers, mobile phones, or virtualized infrastructures. Rather than analyzing given code, we ask what code an intruder could create to break a security goal of the platform. To avoid searching the infinite space of programs that the intruder could come up with (given some initial knowledge) we adapt the lazy intruder technique from protocol verification: the code is initially just a process variable that is getting instantiated in a demand-driven way during its execution. We also take into account that by communication, the malicious code can learn new information that it can use in subsequent operations, or that we may have several pieces of malicious code that can exchange information if they "meet". To formalize both the platform and the malicious code we use the mobile ambient calculus, since it provides a small, abstract formalism that models the essence of mobile code. We provide a decision procedure for security against arbitrary intruder ambients when the honest ambients can only perform a bounded number of steps and without path constraints in communication.

1 Introduction

Mobile Intruder With *mobile intruder* we summarize the problem of executing code from an untrusted source in a trusted environment. The most common example is executing in a web browser code from untrusted websites (e.g., in Javascript). We trust the web browser and surrounding operating system (at least in its initial setup), we have a security policy for executing code (e.g., the Document Object Model in web-browsers), and we want to verify that an intruder cannot design any piece of code that would upon execution lead to a violation of our security policy [10]. There are many similar examples where code from an untrusted source is executed by an honest host such as mobile phones or virtual infrastructures.

Related Problems The mobile intruder problem is in a sense the dual of the *mobile agents* problem where "honest" code is executed by an untrusted environment [3]. The mobile intruder problem has also similarities with the proof-carrying-code (PCC) paradigm [13]. In PCC we also want to convince ourselves

* The research presented in this paper has been partially supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology. The authors thank Luca Viganò for helpful discussions and comments.

that a piece of code that comes from an untrusted source will not violate our policy. In contrast to PCC, we consider here not a concrete given piece of code, but verify that our environment securely executes *every* piece of code. Also, of course, we do *not* require code to be equipped with a proof of its security.

The Problem and a Solution The difficulty to verify a given architecture for running potentially malicious code lies in the fact that there is an infinite number of programs that the intruder can come up with (given some initial knowledge). Even bounding the size of programs (which is hard to justify in general), the number of choices is vast, so that naively searching this space of programs is infeasible.

We observe that this problem is very similar to a problem in protocol verification and that one may use similar verification methods to address it. The similar problem in protocol verification is that the intruder can at any point send arbitrary messages to honest agents. Also here, we have an infinite choice of messages that the intruder can construct from a given knowledge, leading to an infinitely branching transition relation of the system to analyze. While in many cases we can bound the choice to a finite one without restriction [4], the choice is still prohibitively large for a naive exploration.

In order to deal with this problem of large or infinite search spaces caused by the "prolific" intruder, a popular technique in model checking security protocols is a constraint-based approach that we call *the lazy intruder* [11, 12, 14, 5]. In a state where the intruder knows the set of messages K, he can send to any agent any term t that he can craft from this knowledge, written $K \vdash t$. To avoid this naive enumeration of choices, the lazy intruder instead makes a *symbolic* transition where we represent the sent message by a *variable* x and record the constraint $K \vdash x$. During the state exploration, variables may be instantiated and the constraints must then be checked for satisfiability. The search procedure thus determines the sent message x in a demand-driven, lazy way.

A basic idea is now that code can be seen as a special case of a message and that we may use the lazy intruder to lazily generate intruder code for us. There are of course several differences to the problem of intruder-generated message, because code has a dynamic aspect. For instance the code can in a sense "learn" messages when it is communicating with other processes and use the learned messages in subsequent actions. Another aspect is that we want to consider mobility of code, i.e., it may move to another location and continue execution there. We may thus consider that code is bundled with its local data and move together with it, as it is the case for instance on migration operations in virtual infrastructures. As a result, when two pieces of intruder-generate code are able to communicate with each other, then they can exchange all information they have gathered. An example is that an intruder-generated piece of code is able to enter a location, gather some secret information there, and return to the intruder's home base with this information.

Contribution The key idea of this paper is to use the lazy intruder for the malicious mobile code problem: in a nutshell, the code initially written by the

intruder is just a variable x and we explore in a *demand driven, lazy* way what this code could look like more concretely in order to achieve an attack.

Like in the original lazy intruder technique, we do not limit the choice of the intruder, but verify the security for the infinite set of programs the intruder could conceive. Also, like in the lazy intruder for security protocols, this yields only a semi-decision procedure for insecurity, because there can be an unbounded number of interactions between intruder and the environment; this is powerful enough to simulate Turing machines. However bounding the number of steps that honest ambients can perform, we obtain a decision procedure.

For such a result, we need to use a formalism to model the mobile intruder code—or several such pieces of code—and the environment where the code is executed. In this paper we choose the mobile ambient calculus, which is an extension of common process calculi with a notion of mobility of the processes and a concept of boundaries around them, the ambients. The reason for this choice is that we can develop our approach very abstractly and demonstrate how to deal with each fundamental aspect of mobile code without committing to a complex formalization of a concrete environment such as a web-browser running Javascript or the like. In fact, mobile ambients can be regarded as a "minimal" formalism for mobility. Moreover, it has a well-defined semantics which is necessary to formally prove the correctness of our lazy mobile intruder technique. We therefore avoid a lot of technical problems that are immaterial to our ideas, and neither do we tie our approach to one particular application field.

2 The Ground Model

2.1 The Ambient Calculus

We use the ambient calculus as defined by Cardelli and Gordon [7]. There is a basic version and an extension with communication primitives; we present the ambient calculus right away with communication and only mention that our method also works, mutatis mutandis, for the basic ambient calculus. Fig. 1 contains the syntax of the ambient calculus, and Fig. 2 and 3 give the semantics by defining a structural congruence and reduction relation, respectively. In these figures, we have already omitted some primitives that we do not consider in this paper, namely replication, name restriction, and path constraints; we discuss these restrictions in Sec. 2.5.

The ambient calculus is an extension of standard process calculi with the usual constructs 0 for the inactive process, $P \mid Q$ for the parallel execution of processes P and Q, as well as input $(x).P$—binding the variable x in P—and output $\langle M \rangle$. In addition we have a concept of a process running within a boundary, denoted $n[P]$, and this boundary has a name n. For instance one may model by $m[Pv_1[R] \mid v_n[Q]]$ a situation where a process is running on a physical machine m together with virtual machines v_1 and v_2 on which processes R and Q are running, respectively. The communication rule (4) for instance says that processes can only communicate when they run in parallel, but not when they

$P, Q ::=$		processes	$M ::=$		capabilities
	0	inactivity		x	variable
	$P \mid Q$	composition		n	name
	$M[P]$	ambient		$in\ M$	can enter M
	$M.P$	capability action		$out\ M$	can exit M
	$(x).P$	input action		$open\ M$	can open M
	$\langle M \rangle$	output action			

Fig. 1. Considered fragment of the ambient calculus.

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \qquad \frac{P \equiv Q}{P \mid R \equiv Q \mid R}$$

$$\frac{P \equiv Q}{M[P] \equiv M[Q]} \qquad \frac{P \equiv Q}{M.P \equiv M.Q} \qquad \frac{P \equiv Q}{(x).P \equiv (x).Q} \qquad P \mid Q \equiv Q \mid P$$

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid 0 \equiv P$$

Fig. 2. Structural congruence relation.

are separated by ambient boundaries. Process can move with the operations *in* n and *out* n according to rules (1) and (2); also one process can remove the boundary $n[\cdot]$ of another parallel running ambient by the action *open* n according to rule (3). In all positions where names can be used, we may also use arbitrary capabilities M, e.g., one may have strange ambient names like *in in* n, but this is merely because we do not enforce any typing on the communication rules, and we will not consider this in examples.

We require that in all ambients where two input actions $(x).P$ and $(y).P$ occur, different variable symbols $x \neq y$ are used. This is not a restriction since we do not have the replication operator and can therefore make all variables disjoint initially by α-renaming.

2.2 Transition Relation

The definition of the reduction relation $\rightarrow$ in Fig. 3 is standard, however there is a subtlety we want to point out that is significant later when we go to a symbolic relation $\Rightarrow$. The point is that to be completely precise, the symbols n, m, P, Q, R, P', and Q' in these rules are *meta-variables* ranging over names and ambients, respectively. When applying a rule, these variables are supposed to be *matched* with the ambient they are applied to.

To work with the symbolic approach later more easily, let us reformulate this and make explicit the matching by interpreting rules as *rewriting* rules. In this view, the rules (1)–(4) of Fig. 3 define the essential behavior of the *in*, *out*, and

$$n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R] \quad (1)$$

$$m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R] \quad (2)$$

$$open\ n.P \mid n[Q] \to P \mid Q \quad (3)$$

$$(x).P \mid \langle M \rangle \to P\{x \mapsto M\} \quad (4)$$

$$\frac{P' \equiv P \quad P \to Q \quad Q \equiv Q'}{P' \to Q'} \qquad \frac{P \to Q}{n[P] \to n[Q]} \qquad \frac{P \to Q}{P \mid R \to P \mid Q}$$

Fig. 3. Reduction relation of the ambient calculus

open operators and communication, while the other rules simply tell us to which *subterms* of an ambient the rules may be applied. For instance, the ambient $M.P$ does not admit a reduction, even if the subterm P does. We can capture that by an *evaluation context* defined as follows:

$C[\cdot]$	::=	context
	$\cdot$	empty context
	$C[\cdot] \mid P$	parallel context
	$M[C[\cdot]]$	ambient context

We define that each rule $r = L \to R$ of the first four rules of Fig. 3 (where the ambients L and R have free (meta-) variables on the left-hand and right-hand side) induces a transition relation on *closed* ambients as follows: $S \to_r S'$ holds iff there is an evaluation context $C[\cdot]$ and a substitution σ for all the variables of r such that $S \equiv C[\sigma(P)]$ and $S' := C[\sigma(R)]$.[1]

2.3 Ground Intruder Theory

We now define how the intruder can construct ambients from a given knowledge K, which is simply a set of *ground* capabilities (i.e. without variables). This model is defined in the style of Dolev-Yao models of protocol verification as the least closure of K under the application of some operators. These operators are encryption and the like for protocol verification, and here they are the following constructors of ambients (written with their arguments for readability):

$$\Sigma_p = \{0\,,\ P \mid Q\,,\ M[P]\,,\ M.P\,,\ \langle M \rangle\,,\ in\ M\,,\ out\ M\,,\ open\ M\}$$

We here leave out the input $(x).P$ because it is treated by a special rule.

Fig. 4 inductively defines the *ground intruder deduction relation* $K \vdash_V T$ where K is a set of ground capabilities, T ranges over capabilities and processes, and V is a set of variables such that $V = fv(T)$ the *free variables* of T. We require that the knowledge K of the intruder contains at least one name n_i, so

[1] One may additionally allow here that S' can be rewritten modulo $\equiv$ to match the rules of Fig. 3 precisely, but it is not necessary because when applying further transition rules, this is done modulo $\equiv$.

$$\frac{}{K \vdash M}\ M \in K \quad \text{(Axiom)} \qquad \frac{K \vdash P \quad P \equiv Q}{K \vdash Q}\ \text{(Str.Cong.)}$$

$$\frac{K \vdash_{V_1} T_1 \quad \ldots \quad K \vdash_{V_n} T_n}{K \vdash_{\cup_{i=1}^{n} V_i} f(T_1, \ldots, T_n)}\ f \in \Sigma_p \quad \text{(Public Operation)}$$

$$\frac{}{K \vdash_{\{x\}} x}\ x \in \mathcal{V} \quad \text{(Use variables)} \qquad \frac{K \vdash_V P}{K \vdash_{V \setminus \{x\}} (x).P}\ \text{(Input)}$$

Fig. 4. Ground intruder deduction rules.

the intruder can always say *something*. For $V = \emptyset$ we also write simply $K \vdash T$. Let $\mathcal{V}$ denote the set of all variable symbols. The (Axiom) and (Str.Cong.) express that the derivable terms contain all elements of the knowledge K and are closed under structural congruence. The (Public Operation) rule says that derivability is closed under all the operators from Σ_p; here the free variables of the resulting term are the union of the free variables of the subterms. The rule (Use variables) and (Input) together allow the intruder to generate processes that read an input and then use it.

As an example, given intruder knowledge $K = \{in\ n, m\}$ we can derive for instance $K \vdash m[(x).in\ n.out\ x.\langle open\ m\rangle]$.

We use the common term "ground intruder" and later "ground transition system" from protocol verification, suggesting we work with terms that contain no variables. We allow the intruder however to create processes like $(x).P$ where P may freely contain x, and only require that the intruder processes at the end of the day are *closed* terms (without free variables). We may thus correctly call it "closed intruder" and "closed transition system" but we prefer to stick with the established terms.

2.4 Security Questions

We are now interested in security questions of the following form: given an honest ambient and a position within that ambient where the intruder can insert some *arbitrary* code that he can craft from his knowledge, can he break a security goal of the honest ambient? This is made precise by the following definition:

Definition 1. *Let us specify security goals via a predicate attack(P) that holds true for an ambient P when we consider P to be successfully attacked. We then also call P an* attack state. *Let $C[\cdot]$ be an (evaluation) context without free variables that represents the honest ambients and the position where the intruder can insert code. Let finally K_0 be a set of ground capabilities. Then the question we want to answer is whether there exist P_0 and P such that $K_0 \vdash P_0$, $C[P_0] \to^* P$ and attack(P).*

We generalize this form of security questions as expected to the case where the intruder can insert several pieces of code $P_0, \ldots, P_k$ in different locations, and they are generated from different knowledges $K_0, \ldots, K_k$, respectively.

There are many ways to define security goals for ambients, and we have opted here for state-based safety properties rather than observational equivalences. In fact, the most simple goal is that no intruder ambient may ever learn a secret name s. On the ground level, there is a technical difficulty to define $attack(P)$ for secrecy of s, because one needs to keep track of what the intruder code has learned. (For instance one can require the intruder to produce a term of the form $C[s]$ for some context $C[\cdot]$ that is never produced by an honest ambient.) On the symbolic level we define below, the knowledge of intruder processes is obvious in the notation, so that secrecy goals are straightforward to specify then.

Another goal is that the intruder code cannot reach a given position in the ambient. This can be reduced to a secrecy goal—at the destination waits a process that writes out a secret. A more complex goal is containment: a sandbox may host an intruder code and give that code some secret s to compute with, but the intruder code should not be able to get s out of the sandbox. This can again be reduced to secrecy (of another value s') if outside the sandbox a special ambient $s[open\ n_i.\langle s'\rangle]$ is waiting. From this ambient an intruder process (who initially knows the name n_i) can obtain secret s' if it was able to learn s and get out of the sandbox.

Example 1. As an example let us consider the firewall example from [7]:

$$Firewall \equiv w[k[out\ w.in\ k'.in\ w] \mid open\ k'.open\ k''.\langle s\rangle]$$

The goal is that the firewall can only be entered by an ambient that knows the three passwords k, k', and k'' (in fact having capability $open\ k$ instead of k is sufficient). Here the ambient $k[\cdot]$ acts as a pilot that can move out of the firewall, fetch a client ambient (that needs authenticate itself) and move it into the firewall. Suppose we run $Firewall \mid P$ for some ambient P that the intruder generated from knowledge K and define as an attack, if the intruder ambient learns secret s. If K includes $open\ k, k', k''$, then we have an attack, since the intruder can generate the ambient $P \equiv k'[open\ k.k''[(x).P_0]]$ from K. An attack is reached as follows:

$$\begin{aligned}
&Firewall \mid P\\
&\to w[open\ k'.open\ k''.\langle s\rangle] \mid k[in\ k'.in\ w] \mid P\\
&\to w[open\ k'.open\ k''.\langle s\rangle] \mid k'[k[in\ w] \mid open\ k.k''[(x).P_0]]\\
&\to w[open\ k'.open\ k''.\langle s\rangle] \mid k'[in\ w \mid k''[(x).P_0]]\\
&\to w[open\ k'.open\ k''.\langle s\rangle \mid k'[k''[(x).P_0]]]\\
&\to w[\langle s\rangle \mid (x).P_0]\\
&\to w[P_0[x \mapsto s]]
\end{aligned}$$

If the knowledge K from which the intruder ambient is created does not include $open\ k$ (or k), k' and k'', then no attack is possible. Also if we rather define the attack to get the secret s out of the firewall again, the attacker cannot do it unless he knows w (which is a secret of the firewall), as an example for containment.

2.5 The Considered Fragment

For the automation, we have made some restrictions w.r.t. the original ambient calculus. The replication operator $!P \equiv P \mid !P$ together with the creation of

new names allows for simulating arbitrary Turing machines and thus prevents a decision procedure for security. Similar to the lazy intruder in protocol verification, we thus bound the steps that honest ambients can perform and do this by simply disallowing the replication operator for honest ambients. Without replication, one of the main reasons for the name restriction operator $\nu n.P$ is gone, since we can α-rename all restricted names so that they are unique throughout the ambients. Note that the name restriction is also useful for goals of observational equivalence, which are essential for privacy goals [1, 2] but which we do not consider in this paper.

Note that we do not bound the size of ambients that the intruder creates: the derivation relation $K \vdash P$ allows him to make arbitrary use of all constructors. It may appear as if the intruder were bounded because $K \vdash P$ does not include the replication operator either, but this is not true: an attack always consists of a finite number of steps (as violation of a safety property) and thus every attack that can be achieved by an intruder ambient with replication can be achieved by one without replication (just by "unrolling" the replication as much as necessary for the particular attack). The difference between unbounded intruder ambients and bounded honest ambients thus stems from the fact that we ask questions of the form: "can a concrete honest ambient (of fixed size) be attacked by *any* dishonest ambient (of arbitrary size)?"

We do not need give the intruder the ability to create arbitrary new names. The reason is that we have no inequality checks in the ambient calculus, i.e., no ambient can check upon receiving a capability n that it is different from all names it knows (e.g. to prevent replays). Thus, whatever attack works when the intruder uses different self-created names works similarly with always using the same intruder name n_i that we give the intruder initially.

Finally, the extension of ambients with communication includes so-called *path constraints* of the form $M.M'$ that can be communicated as messages. Note that this is not ordinary concatenation of messages (which the symbolic techniques we use can easily handle) but sequences of instructions and only after the first has been successfully executed, the next one becomes available, and so the paths cannot be decomposed. Since this includes several problems that would complicate our method, we have excluded them.

3 Symbolic Ambients

We now introduce the symbolic, constraint-based approach that is at the core of this paper. To efficiently answer the kind of security questions we formalized in the previous paragraph, we want to avoid search the space of all ambients that an intruder can come up with. To that end, we use the basic idea of the symbolic, constraint-based approach of protocol verification, also known as the lazy intruder [11, 12, 14, 5].

When an agent in a protocol wants to receive a message of the form t—a term that contains variables—we avoid enumerating the set of all messages that the intruder can generate and that are instances of t (because this set is often

very large or infinite). Rather we remember the constraint $K \vdash t$ where K is the set of messages that the intruder knows at the point when he sends the instance of t. We then proceed with states that have free variables, namely the variables of t (and of other messages as they sent and received). The allowed values for these variables is governed by the constraints. For a fixed number of agents and sessions, this gives us a symbolic finite-state transition system. What remains to obtain a decision procedure (given a characterization of attack states) is a decision procedure for satisfiability of the constraints attached to each state. The complexity of this decision procedure has been studied for a variety of algebraic theories of the operators involved, e.g. [8, 9]; in the easiest theory, the free algebra, the problem is NP-complete [14]. Since one can check satisfiability of the constraints on-the-fly (and prune the search tree once a state has unsatisfiable constraints), messages during the search get successively instantiated with more concrete messages—following what the transitions require in a demand-driven, lazy way. Hence the name.

Now we carry over this idea to ambients and apply it to the ambients that were written by the intruder, i.e. lazily creating the intruder-generated ambients during the search. Recall that in the previous section we defined security problems as reachability of an attack state from $C[P_0]$ where $C[\cdot]$ is a given honest agent and $K_0 \vdash P_0$ is any intruder process generated from a given initial knowledge K_0. We thus could thus simply work with a symbolic state $C[P_0]$ where P_0 is a variable and we have the constraint $K_0 \vdash P_0$. We then have to define appropriate transition rules for these *symbolic* ambients.

One inconvenience attached to using a variable to represent a process is that with every step the process changes and we need to then introduce new variables and relate them to the old ones. Moreover the processes can learn new information by communication with others, so the available knowledge changes. For these two reasons we follow the most convenient option and simply represent an intruder generated process by $\boxed{K}$ where K is the knowledge from which it was created. Here K is a set of capabilities and intuitively $\boxed{K}$ represents *any* process that can be created from K. If an ambient contains two occurrences of $\boxed{K}$ for the same K, they may represent different processes. K may contain variables because we will also handle the communication between ambients with the lazy intruder technique. We thus extend the syntax of ambients P, Q of Fig. 1 by $\boxed{K}$, and we consider symbolic security problems as reachability of a symbolic attack state (defined below) from an initial state $C[\boxed{K}]$ where $C[\cdot]$ is an honest ambient environment that the intruder code is running in.

A symbolic processes will also be equipped with constraints which have the following syntax:

$$
\begin{array}{lll}
\phi, \psi ::= & & \text{constraints} \\
& K \vdash M & \text{intruder deduction constraint} \\
& x = M & \text{substitution} \\
& \phi \wedge \psi & \text{conjunction}
\end{array}
$$

Intuitively, $K \vdash M$ means that capability/message M can been generated by the intruder from knowledge K. In fact, we do not need $K \vdash P$ for processes P in the symbolic constraints, due to the $\boxed{K}$ notation and since we have no construct for sending processes.

Semantics We define the semantics for pairs (S, ϕ) of symbolic ambients and constraints as a (usually infinite) set of closed processes. An *interpretation* $\mathcal{I}$ is a mapping from all variables to ground capabilities. We extend this to a morphism on capabilities, processes, and sets of processes as expected; where $\mathcal{I}$ substitutes only free occurences of variables. We define the model relation as follows:

$$\begin{aligned} \mathcal{I} \models K \vdash M &\text{ iff } \mathcal{I}(K) \vdash \mathcal{I}(M) \\ \mathcal{I} \models x = M &\text{ iff } \mathcal{I}(x) = \mathcal{I}(M) \\ \mathcal{I} \models \phi \wedge \psi &\text{ iff } \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \psi \end{aligned}$$

The semantics of (S, ϕ) is the set of possible instantiation of all variables and intruder code pieces $\boxed{K}$ with closed processes:

$$\begin{aligned} [\![P, \phi]\!] &= \{Q \mid \mathcal{I} \models \phi \wedge Q \in ext(\mathcal{I}(P))\} \\ ext(\boxed{K}) &= \{P \mid K \vdash P\} \\ ext(x) &= \{x\} \\ ext(n) &= \{n\} \\ ext(f(T_1, \ldots, T_n)) &= \{f(T_1', \ldots, T_n') \mid T_1' \in ext(T_1) \wedge \ldots \wedge T_n' \in ext(T_n)\} \end{aligned}$$

Here the T_i range over capabilities and processes and f ranges over all constructors of capabilities and processes. Note the case $ext(x)$ can only occur when processing a subterm of an ambient where x is bound, so no free variables occur in any $S_0 \in [\![P, \phi]\!]$.

Symbolic Attack States Since each intruder process $\boxed{K}$ keeps track of its knowledge (it is extended when the intruder processes comunicates with another process and learns a capability), we can formalize secrecy of a name s easily on the symbolic level: $attack(S, \phi)$ shall hold if S contains an intruder process $\boxed{K}$ such that $\phi \wedge K \vdash s$ is satifiable.

Lazy Intruder Constraint Reduction A decision procedure for satisfiability of $K \vdash M$ constraints can be designed straightforwardly in the style of [12, 5], since we just need to handle the constructors for capabilities, namely *in*, *out*, and *open*, and we have no destructors (or algebraic properties). Note that these approaches require a well-formedness conditions on the constraints, namely that variables originate only from an intruder choice and that the intruder knowledge grows monotonically. We need a small adaption for our setting, because in general we have several intruder processes with incomparable knowledge; however the knowledge in each of these processes grows monotonically. Thus, we define well-formedness by the existence of a *partial order* (instead of a total order) along which the intruder knowledge grows monotonically and along which variables first occur on the right-hand side of a constraint.

3.1 Symbolic Transition Rules

We now define a symbolic transition relation on symbolic processes with constraints of the form $(S, \phi) \Rightarrow (S', \phi \wedge \psi)$. Note that the constraints are *augmented* in every step, i.e., all previous constraints ϕ remain and new constraints ψ may be added.

We first want to lift the standard transition rules on ground ambients of Section 2.2 to the symbolic level. The idea is to replace the rule *matching* defined above with rule *unification*. Recall that before, we have essentially defined a transition rule $r = L \to R$ to be applicable to state S if $S = C[\sigma(L)]$ for some substitution σ and evaluation context $C[\cdot]$. For the symbolic level we have that S may contain free variables that need to be substituted as well. Suppose the rule r does not contain any variables that occur in the symbolic state (S, ϕ) (which is achieved by α-renaming the rule variables). Thus define that $(S, \phi) \Rightarrow_r (S', \phi \wedge \psi)$ holds iff there is an evaluation context $C[\cdot]$ and a term T such that:

- $S \equiv C[T]$;
- σ is a most general unifier of T and L modulo $\equiv$, i.e., $\sigma(T) \equiv \sigma(L)$ and for no generalization τ of σ it holds that $\tau(T) \equiv \tau(L)$; and
- $S' = \sigma(C[\sigma(R)])$ and $\psi = eq(\sigma)$.

Observe σ may now replace also variables that occur in S and thus σ is applied also to $C[\cdot]$.

Example 2. Using the *in* rule, we can now make the following symbolic transition: $(x[P] \mid y[in\ z.Q], \phi) \Rightarrow (z[P \mid y[Q]], \phi \wedge x = z)$

Similarly, also $(x[P] \mid y[in\ z.\boxed{K}], \phi) \Rightarrow (z[P \mid y[\boxed{K}]], \phi \wedge x = z)$ is possible for an intruder generated piece of code $\boxed{K}$.

So far, however, the rules do not allow us to make an *in* transition on the following state: $(x[P] \mid y[\boxed{K}], \phi)$ even if the intruder can generate a process of the form $in\ z.Q$ from knowledge K. We will see below how to add appropriate rules for intruder-generated ambients, so that for instance in the above state an in-rule is applicable.

It is thus clear that the described symbolic transitions are sound (i.e., all states that are reachable in the symbolic model represent states that are reachable in the standard ground model) but not yet complete. More precisely, in the condition $S \equiv C[T]$ above we restrict the application of rule r to contexts that exist in S—without first instantiating intruder code like $\boxed{K}$ first. To give a complete set of rules for intruder ambients is the subject of the rest of this section.

Intruder-written Code We now come to the very core of the approach: lazily instantiating a piece of code $\boxed{K}$ that the intruder generated from knowledge K, with a more concrete term in a demand-driven way. It is basically what is lacking after the lifting of the ground rules that we have just described, namely

when an "abstract" piece of intruder-written code $\boxed{K}$ prevents the application of a rule that would be applicable when replacing $\boxed{K}$ with some more concrete process P for which $K \vdash P$. Obviously we would like to identify such situations without enumerating all processes P that can be generated from K.

In the example $x[P] \mid y[\boxed{K}]$ we discussed above, we have the following possibility: if the intruder code marked $\boxed{K}$ were to have the shape $in\ x.Q$, we could apply the in rule and get to the state $x[y[\boxed{K}] \mid P]$, assuming $K \vdash in\ x$. Note the residual code (inside $y[\cdot]$ after the move) is again something generated from knowledge K.

There is a systematic way to obtain all rules that are necessary to achieve completeness, namely by answering the following question: given a symbolic ambient with constraints (S, ϕ), any ground ambient $S_0 \in [\![S, \phi]\!]$, and a transition $S_0 \to S_0'$ what rule do we need on the symbolic level to perform an analogous transition? Thus, we want to reach an $(S', \phi \wedge \psi)$ (in zero or more steps) such that $S_0' \in [\![(S', \phi \wedge \psi)]\!]$. Of course, the rule should also be sound (i.e. all $S_0' \in [\![(S', \phi \wedge \psi)]\!]$ are reachable with ground transition rules from some $S_0 \in [\![(S, \phi)]\!]$). Soundness is relatively easy to see, because we need to consider rules only in isolation. We now systematically derive rules for each case of (S, ϕ), S_0, and S_0' that can occur and thereby achieve a sound and complete set of symbolic transition rules.

Recall that by the definition, for a transition from S_0 to S_0' with rule $r = L \to R$, we need to have an evaluation context $C_0[\cdot]$ and a substitution σ of the rule variables such that $S_0 = C[\sigma(L)]$ and $S_1 = C[\sigma(R)]$.

The symbolic transition rules we have defined above already handle the case that the symbolic state S has the form $S = C'[T]$ where $\sigma(C')[\cdot] = C[\cdot]$ and $\sigma(L) \equiv \sigma(T)$ (as shown in the examples previously) where at a corresponding position a similar rule (under renaming) can be applied without instantiating intruder code. This includes the case that a rule variable P of type ambient is unified with a piece $\boxed{K}$ of intruder code.

Another case that does not require further work is when the rule match in S_0 is for a subterm of intruder-generated code, i.e. where we have $\boxed{K}$ in the symbolic term S. Here we use the fact that intruder deduction is closed under evaluation: if $K \vdash P$ and $P \to P'$, then also $K \vdash P'$.

Therefore all remaining cases that we need to handle are where one or more proper subterms of the redex $\sigma(L)$ in S_0 are intruder-written code that are not trivial, i.e. represent a variable in L. We make a case-split

- by the different transition rules for $\to$, namely (1)–(4),
- and by how S relates to matching subterm of the transition rule in S_0.

In-Rule Let us mark three positions in the in rule which could be intruder-written code and that are not yet handled:

$$\overbrace{n[\underbrace{in\ m.P}_{p2} \mid Q]}^{p1} \mid \underbrace{m[R]}_{p3} \to m[n[P \mid Q] \mid R]$$

In fact, this notation contains a simplification: for instance looking at position p_2, we could also have the variant that the intruder code is of the form $in\ m.P \mid P'$. In such a case, the intruder code piece $\boxed{K}$ in the symbolic state would not exactly correspond to a subterm of the matched rule, but only after "splitting" $\boxed{K}$ into $\boxed{K} \mid \boxed{K}$. Such a splitting rule would obviously be sound, but we do not want to include it, and rather perform such splits only in a demand driven way (as the following cases show)—and to keep the notation simple for the positions in the rules. So all positions indicated here are considered under the possibility that the intruder code itself is a parallel composition; note also we are matching/unifying modulo $\approx$.

In rule with intruder code at position p_1 The first case we consider is when only the p_1 rule is intruder code, i.e., we have some intruder code running in parallel with an ambient $m[R]$; then the intruder code may be able to enter m if it has the capability $in\ m$. As said before, we could have the case that the intruder code first splits into two parts and only one part enters m while the other part stays outside. This can be helpful if the intruder code does not have the capability $out\ m$. Since the intruder code can always be trivially 0 if there is nothing to do, it is not a restriction to make the split, so we avoid two rules. We obtain:

$$\boxed{K} \mid m[R] \Rightarrow \boxed{K} \mid m[x[\boxed{K}] \mid R] \text{ and } \psi = K \vdash in\ m \wedge K \vdash x \qquad (5)$$

Here we denote with ψ the new constraints that should be added to the symbolic successor state. x is a new variable symbol (that does not occur so far). The reason for introducing this new symbol x is that a process cannot move without being surrounded by an ambient $n[\cdot]$ construct; as the $n[\cdot]$ of the normal in rule has now become part of the $\boxed{K}$ code, we need to say that the intruder himself created the ambient. As there is no obligation to pick a particular name for that ambient, we simply leave it open and just require it is one the intruder can construct from knowledge K. Note that would be unsound in general to simplify the right-hand side to $m[\boxed{K} \mid R]$ because the intruder cannot get rid of the surrounding $x[\cdot]$ (even though self-chosen) without another process performing $open\ x$.

To see the soundness of this rule, consider that the intruder code matched on the left-hand side of the rule should have the form $P \mid x[in\ m.Q]$ for some processes P and Q generated from knowledge K. These are then represented by the two $\boxed{K}$ pieces on the right-hand side of the rule.

In rule with intruder code at position p_2 Here, intruder code is running inside ambient n that runs in parallel with ambient m. The intruder code can move ambient n into m, if it has the capability $in\ m$:

$$n[\boxed{K} \mid Q] \mid m[R] \Rightarrow m[n[\boxed{K} \mid Q] \mid R] \text{ and } \psi = K \vdash in\ m \qquad (6)$$

Note that we could have again the situation that the intruder code is a parallel composition, i.e. of the form $in\ m.P_1 \mid P_2$. However, then after the move we still have $P_1 \mid P_2$ and we thus do not make the split explicit, because this case is still subsumed by $\boxed{K}$ on the right-hand side.

In rule with intruder code at position p_3 Now we consider the situation that an honest ambient $n[in\ m.P \mid Q]$ that wants to enter ambient m runs in parallel with intruder code. If the intruder code has name m, it can provide the ambient that the honest process can then enter:

$$n[in\ m.P \mid Q] \mid \boxed{K} \Rightarrow m[n[P \mid Q] \mid \boxed{K}] \mid \boxed{K} \text{ and } \psi = K \vdash m \qquad (7)$$

Here we have again an explicit split of the intruder process into two parts. This is because the concrete intruder process that is partially matched by the left-hand side may have the form $m[R_1] \mid R_2$, i.e. not entirely running within m, and we thus need to denote that residual process explicitly on the right-hand side.

In rule with intruder code at several positions If the intruder code is at several positions of the rule, we get the following situations. Obviously we do not need to consider the combination $(p_1) + (p_2)$ because (p_2) is a subposition of (p_1). The case $(p_1) + (p_3)$ means that we have two intruder processes (in general with different knowledge) to run in parallel: $\boxed{K} \mid \boxed{K'}$. We will show below (when we treat communication) that what they can achieve together is to pool their knowledge and join to one process $\boxed{K \cup K'}$.

What is left is the combination $(p_2) + (p_3)$ which means that one intruder process runs inside an ambient n and that runs in parallel with another intruder process. This case we can express by the following rule:

$$n[\boxed{K} \mid Q] \mid \boxed{K'} \Rightarrow x[n[\boxed{K} \mid Q] \mid \boxed{K'}] \mid \boxed{K'} \text{ and } \psi = K \vdash in\ x \wedge K' \vdash x$$

Note that the two processes that we start with may not have the same knowledge (here K and K'). Again, we have an explicit split on the side of the K'-generated process into a part that is entered by $n[\cdot]$ and one that remains outside. Also, again, this rule has a new variable for any ambient name x that can be entered; this name needs to be part of K' while K only needs to have the $in\ x$ capability.

The rule in this form is an obstacle for termination of our approach. Observe that the left-hand side ambient $n[\cdot]$ occurs identically as a subterm on the right side; so the rule "packs in" the $n[\cdot]$ ambient into another $x[\cdot]$ ambient. While for other rules we can limit the repeated application (as in the termination proof below) it is here easier to avoid the arbitrary packing. To achieve this without loosing completeness, observe that the additional $x[\cdot]$ layer is only relevant if some honest process comes into contact that wants to perform an $in\ x$, $out\ x$, or $open\ x$. Since $K' \vdash x$ these cases can be handled by other symbolic rules. The case of out can only occur if it is part of the honest process Q i.e., $Q = (out\ x.R) \mid R'$ and the outer process $\boxed{K'}$ has x. The resulting rule for this special case is much easier to handle:

$$\begin{aligned} &n[\boxed{K} \mid (out\ x.R) \mid R'] \mid \boxed{K'} \Rightarrow n[\boxed{K} \mid R \mid R'] \mid \boxed{K'} \\ &\text{and } \psi = K \vdash in\ x \wedge K' \vdash x \end{aligned} \qquad (8)$$

Out Rule For the *out* rule we have two positions of intruder code to consider:

$$m[\overbrace{n[\underbrace{out\ m.P}_{p2} \mid Q]}^{p1} \mid R] \to n[P \mid Q] \mid m[R]$$

Out Rule with intruder code at position p_1 Here we have the situation that the intruder code is within an ambient m and has the capability *out* m. To move parts of the code, the intruder must put it within some ambient x (where x is again a new variable symbol):

$$m[\boxed{K} \mid R] \Rightarrow x[\boxed{K}] \mid m[\boxed{K} \mid R] \text{ and } \psi = K \vdash out\ m \wedge K \vdash x \tag{9}$$

Out rule with intruder code at p_2 This situation is similar except that the intruder code is already contained within an ambient n. We then have:

$$m[n[\boxed{K} \mid Q] \mid R] \Rightarrow n[\boxed{K} \mid Q] \mid m[R] \text{ and } \psi = K \vdash out\ m \tag{10}$$

This subsumes also the case that there is intruder code at both in m and in n (i.e. also within what is matched as R here).

Open-Rule The open rule has also just two positions for intruder code, the opening code and the opened code:

$$\underbrace{open\ n.P}_{p1} \mid \underbrace{n[Q]}_{p2} \to P \mid Q$$

The rules for the intruder code at p_1 and at p_2, respectively are immediate:

$$\boxed{K} \mid n[Q] \Rightarrow \boxed{K} \mid Q \text{ and } \psi = K \vdash open\ n \tag{11}$$

$$open\ n.P \mid \boxed{K} \Rightarrow P \mid \boxed{K} \text{ and } \psi = K \vdash n \tag{12}$$

The case $(p1) + (p2)$ is again the case of two parallel communicating processes that is treated next.

Communication Rule Again there are two possible positions where intruder code could reside, namely as the sender or as the receiver:

$$\underbrace{(x).P}_{p1} \mid \underbrace{\langle M \rangle}_{p2} \to P[x \mapsto M]$$

Communication with the intruder receiving The intruder can receive a message M from an honest process running in parallel:

$$\boxed{K} \mid \langle M \rangle \Rightarrow \boxed{K \cup \{M\}} \tag{13}$$

Here the resulting intruder process has the message M simply added to its knowledge. The idea is that the remaining process can behave like any process that the intruder could have created, if he initially knew $K \cup \{M\}$. To see that this is sound, consider that the intruder process would have the form $(x).P$ for a new variable x that can occur arbitrarily in P. Thus if this process reads M, the resulting $P[x \mapsto M]$ is is a process that can be generated from knowledge $K \cup \{M\}$ if P was created from knowledge K.

Communication with the intruder sending For the case that intruder code sends out a message that is received by an honest ambient, we can be *truly lazy*:

$$(x).P \mid \boxed{K} \Rightarrow P \mid \boxed{K} \text{ and } \psi = K \vdash x \tag{14}$$

Here, we do not instantiate the message x that is being received, we simply add the constraint that x must be something the intruder can generate from knowledge K. This is in fact the classic case of the lazy intruder—postponing the choice of a concrete message that one sends to an agent. Since the intruder knowledge contains at least one name, there is always "something to say", but what it is will only be determined if the variable x gets unified later on for applying some rule (which can render the $K \vdash x$ constraint unsatisfiable).

Communication with the intruder both sending and receiving Finally we have the rule that was mentioned above already: when two intruder processes meet they can exchange their knowledge and work together further on:

$$\boxed{K} \mid \boxed{K'} \Rightarrow \boxed{K \cup K'} \tag{15}$$

This is sound because every $k \in K \setminus K'$ can be sent from the first to the second process until we have $\boxed{K} \mid \boxed{K \cup K'}$ and then the second part subsumes the first, so we can simplify it to $\boxed{K \cup K'}$. Observe that this rule can also be used when we restrict ourselves to the pure ambient calculus without communication: we then simply have two processes in parallel with capabilities K and K', respectively, and what they can achieve is anything a process with capabilities $K \cup K'$ can achieve (even without communication).

We have systematically developed rules for the symbolic transition system, so that every possible transition $\rightarrow$ of the original (ground) transition is captured by some symbolic transition $\Rightarrow$ and all given rules are sound, thus we have:

Theorem 1. *Given a symbolic ambient and constraint (S, ϕ) then*

- *any symbolic successor state represents only ground states that are indeed reachable from $[\![S, \phi]\!]$ (soundness): $\{S'_0 \mid \exists S', \psi.(S, \phi) \Rightarrow (S', \phi \wedge \psi) \wedge S'_0 \in [\![(S', \phi \wedge \psi)]\!]\} \subseteq \{S'_0 \mid \exists S_0 \in [\![S, \phi]\!] \wedge S_0 \rightarrow^* S'_0\}$*
- *any successor of a ground state from $[\![S, \phi]\!]$ is indeed covered by a reachable symbolic state (completeness): $\{S'_0 \mid \exists S', \psi.(S, \phi) \Rightarrow^* (S', \phi \wedge \psi) \wedge S'_0 \in [\![(S', \phi \wedge \psi)]\!]\} \supseteq \{S'_0 \mid \exists S_0 \in [\![S, \phi]\!] \wedge S_0 \rightarrow S'_0\}$*

Together with the following theorem, we achieve a terminating a correct search procedure for the reachability of an attack state:

Theorem 2. *The symbolic transition relation is finitely branching and in an infinite run* $(S_0, \phi_0) \Rightarrow (S_1, \phi_1) \Rightarrow \ldots$ *we can effectively recognize an* n *after which all* (S_i, ϕ_i) *are semantically equal to some previous state* (S_j, ϕ_j).

Proof. We partition the reduction rules into three categories.

Category Consume Rule These rules are characterized by "consuming" part of an honest process and therefore cannot be applied ad infinitum; this category consists of the standard rules (1)–(4) as well as (7),(8), (11), (12), (13), (14).

Category Fixedpoint Rule These rules are characterized by the intruder process "spreading" into another location without "giving up" its current position. These are the rules (5) and (9). When two intruder processes meet by these fixedpoint rules, they can share their knowledge. We can give a finite upper-bound for the intruder knowledge—all names and all capabilities that can ever be communicated by honest processes. Also the number of locations (that are not created by the intruder himself) is bounded. Therefore there is a fixedpoint for what these rules can allow the intruder to reach. Note that the introduction of variables x in the rules (5) and (9) means that the intruder chooses some name from his knowledge and he can in principle do that again and again (with a new variable), producing terms of the form $x_1[\boxed{K}] \mid x_2[\boxed{K}] \mid \ldots$. These can actually all merge to $x_1[\boxed{K}]$ since $K \vdash x_1 \wedge K \vdash x_2 \wedge \ldots$. This is not a restriction, because if any of the x_i ever gets instantiated to a concrete name (which must still be contained in K), we can just reproduce again a generic $x_2[\boxed{K}]$ from it using rule (9). Thus, it is not a restriction to apply rules (5) and (9) if there is not yet a generic $x[\boxed{K}]$ at the destination of the move with the same or greater knowledge. (Here *generic* means that $K \vdash x$ and x is not further constrained.)

Category Loop Rule The last category are rules that potentially allow for loops. This category comprises (6) and (10). For instance $n[\boxed{\{in\ m, out\ m\}}] \mid m[P] \Leftrightarrow m[n[\boxed{\{in\ m, out\ m\}}] \mid P]$ gives rise to an infinite loop without consuming honest processes or reaching a fixedpoint. However the rules (6) and (10) do not change the size of the ambient but just let an ambient move. Thus, they eventually produce no new states in the search.

Suppose now that we have an infinite run without state repetition, then only in an finite initial portion of the run, we can have applications of consume and fixedpoint rules, so we need an infinite application of the looping rules without state repetition which is absurd. □

3.2 Examples

Let us reconsider the firewall example from before, and see how a lazy intruder ambient would find the attack. In contrast to the original specification, we leave

open how the intruder ambient P exactly works, and rather specify that it is some process generated from the initial knowledge $K = \{in\ k, k', k''\}$:

$$
\begin{array}{ll}
(\mathit{Firewall} \mid \boxed{K}, \mathit{true}) & \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k[\mathit{in}\ k'.\mathit{in}\ w] \mid \boxed{K}, \mathit{true}) & \text{by rule (2)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[k[\mathit{in}\ w] \mid \boxed{K}] \mid \boxed{K}, \phi_1) & \text{by rule (7)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[\mathit{in}\ w \mid \boxed{K}] \mid \boxed{K}, \phi_2) & \text{by rule (11)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle \mid k'[\boxed{K}]] \mid \boxed{K}, \phi_2) & \text{by rule (1)} \\
\Rightarrow (w[\mathit{open}\ k''.\langle s\rangle \mid \boxed{K}] \mid \boxed{K}, \phi_2) & \text{by rule (3)} \\
\Rightarrow (w[\langle s\rangle \mid \boxed{K}] \mid \boxed{K}, \phi_3) & \text{by rule (12)} \\
\Rightarrow (w[\boxed{K \cup \{s\}}] \mid \boxed{K}, \phi_3) & \text{by rule (13)}
\end{array}
$$

where we have collected the constraints $\phi_1 = K \vdash k'$, $\phi_2 = \phi_1 \wedge K \vdash \mathit{open}\ w$, and $\phi_3 = \phi_2 \wedge K \vdash k''$. These constraints are obviously satisfiable iff K includes $\mathit{open}\ k$ (or k), k', and k''. This corresponds to the attack we had described on the ground model, only here we found it *lazily* during the search, rather than specifying the process up front. Actually, one needs to reverse engineer the intruder's recipe from the attack trace. What is even more obvious to see than in the ground case is the point where the malicious ambient learns the secret s.

Another difference to the original trace is that we have an intruder process $\boxed{K}$ remaining at the top-level the entire time. This is basically reflecting the fact that the intruder process could be a parallel composition of two parts only one of which enters the firewall—the position outside the does not have to be "given up" by the intruder.

In the case that learning the secret s alone is not the goal, but to get it out of the firewall. Indeed we can apply rule (11) to the last reached state to get $(\boxed{K \cup \{s\}} \mid \boxed{K}, K \vdash \mathit{open}\ w \wedge \phi_3)$. (Further, using the (15) rule, the two intruder processes can merge again, yielding $\boxed{K \cup \{s\}}$.) The new constraint $K \vdash \mathit{open}\ w$ however is not satisfiable, so this symbolic state has an empty semantics (no attack is realizable in this way) and can be discarded from the search. In fact, there is no reachable symbolic state with satisfiable constraints where the secret s is in an intruder process that is not below $w[\cdot]$.

Ambient in the Middle The previous example has basically identified how an honest client (authenticating itself by the knowledge of the keys k, k', and k'') is supposed to behave, namely $\mathit{Client} \equiv k'[\mathit{open}\ k.k''[\langle s\rangle]]$ where s is again a name that should be secret from the intruder. We now consider the case that such an honest client and firewall execute in the presence of an intruder ambient K:

$$
\begin{array}{ll}
(\mathit{Firewall} \mid \mathit{Client} \mid \boxed{K}, \mathit{true}) & \\
\Rightarrow (\mathit{Firewall} \mid k'[\mathit{open}\ k.k''[\langle s\rangle] \mid \boxed{K}] \mid x[\boxed{K}], \phi_1) & \\
\Rightarrow^2 (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[\mathit{in}\ w \mid k''[\langle s\rangle] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) & \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle \mid k'[k''[\langle s\rangle] \mid x[\boxed{K}]]] \mid \ \mid \ \mid \boxed{K}, \phi_1) & \\
\Rightarrow (w[\langle s\rangle \mid k''[\langle s\rangle] \mid \boxed{K}] \mid \boxed{K}, \phi_2) &
\end{array}
$$

where $\phi_1 = K \vdash in\ k' \wedge K \vdash x$ and $\phi_2 = \phi_1 \wedge K \vdash x = k''$. Thus, the intruder can inject code into the firewall (that is not bounded by $x[\cdot]$ and can obtain s) if he knows only $in\ k'$ and k''. The *open* k capability is not needed, since this is done by the client after the intruder has infected it.

Communication Example As an example where capabilities are communicated consider the process $n_1[\boxed{K_1} \mid n_2[in\ n_3.\langle in\ n_4\rangle]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]]$ where $K_1 = \{n_3, open\ n_2\}$ and $K_2 = \{open\ n_1\}$. Let the goal be that there is no intruder process who will know both *open* n_1 and *open* n_2. The lazy mobile ambient technique finds an attack as follows:

$$
\begin{array}{ll}
(n_1[\boxed{K_1} \mid n_3[\boxed{K_1} \mid n_2[\langle in\ n_4\rangle]]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], true) & \\
\Rightarrow (n_1[\boxed{K_1} \mid n_3[\boxed{K_1} \mid \langle in\ n_4\rangle]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_1) & \text{by rule (11)} \\
\Rightarrow (n_1[\boxed{K_1} \mid n_3[\boxed{K_1 \cup \{in\ n_4\}}]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_1) & \text{by rule (13)} \\
\Rightarrow (n_1[\boxed{K_1} \mid \boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_2) & \text{by rule (11)} \\
\Rightarrow (n_1[\boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_2) & \text{by rule (15)} \\
\Rightarrow (n_1[\boxed{K_1 \cup \{in\ n_4\}}] \mid n_4[\boxed{K_2}] \mid n_5[0], \phi_2) & \text{by rule (2)} \\
\Rightarrow (n_4[\boxed{K_2} \mid n_1[\boxed{K_1 \cup \{in\ n_4\}}]] \mid n_5[0], \phi_3) & \text{by rule (6)} \\
\Rightarrow (n_4[\boxed{K_2} \mid \boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[0], \phi_4) & \text{by rule (11)} \\
\Rightarrow (n_4[\boxed{K_2 \cup K_1 \cup \{in\ n_4\}}] \mid n_5[0], \phi_4) & \text{by rule (15)}
\end{array}
$$

where we the following satisfiable constraints: $\phi_1 = K_1 \vdash open\ n_2$, $\phi_2 = \phi_1 \wedge K_1 \vdash open\ n_3$, $\phi_3 = \phi_2 \wedge K_1 \cup \{in\ m_4\} \vdash in\ m_4$, and $\phi_4 = \phi_3 \wedge K_2 \vdash open\ n_1$. We have reached a state where an intruder ambient knows both *open* n_1 and *open* n_2.

4 Conclusions

We have transferred the symbolic lazy intruder technique from protocol verification to a different problem: an intruder who creates malicious code for execution on some honest platform. This gives us an efficient method to check whether the platform achieves its security goals for *any* intruder code, because we avoid the naive search of the space of possible programs that the intruder can come up with. Instead we determine this code in a demand-driven, lazy way.

Our approach is closest to a model-checking technique. In contrast to static analysis approaches, it works without over-approximation, but requires a bounding of the number of steps that honest agents can perform. The symbolic nature however allows to work without any bound on the size of programs that the intruder can generate. This is similar to the original use of the lazy intruder in protocol verification [11, 12, 14, 5].

We have used a fragment of the mobile ambients calculus with communication as a small and succinct formalism to model both the platform and the mobile code [7]. We have omitted the replication operator in order to bound honest ambients (though not the intruder). We have omitted the path constraints

because they induce considerable complications for our approach and leave their integration for future work. We also plan to consider the extension of boxed ambients introduced by Bugliesi et al. [6] which add interesting means for access control and communication. We also note that our method can easily support the extension of capabilities/messages with the usual operators from (symbolic) protocol verification like encryption and signing.

We believe that the approach we have presented here is generally applicable to the formal analysis of platforms that host mobile code. The key elements can be summarized as follows. First, the code can be lazily developed by exploring at each step which operations can be performed next and what data is needed. This data is handled lazily as well. Second, the intruder code has a notion of knowledge that it can use in further operations and communications, and every received message adds to this knowledge. Third, the code may be able to move to other locations; two pieces of intruder code that meet then pool their knowledge.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *ACM symposium on principles of programming languages*, pages 104–115, 2001.
2. M. Abadi and C. Fournet. Private Authentication. *Theoretical Computer Science*, 322(3):427 – 476, 2004.
3. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, pages 2–11, 2001.
4. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *FSTTCS*, pages 376–387, 2007.
5. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
6. M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: The calculus of boxed ambients. *ACM Trans. Program. Lang. Syst.*, 26(1):57–124, 2004.
7. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
8. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FST TCS'03*, LNCS 2914, pages 124–135, 2003.
9. S. Delaune, P. Lafourcade, D. Lugiez, and R. Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206(2-4):312–351, 2008.
10. T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser model for security analysis of browser-based protocols. In *ESORICS*, pages 489–508, 2005.
11. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols*, 1999.
12. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.
13. G. C. Necula. Proof-carrying code. In *POPL*, pages 106–119, 1997.
14. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.